SecurityDynamics



RSA Data Security
A Security Dynamics Company

Hardware-based Random Number Generation

An RSA Data Security White Paper



RSA Data Security, Inc.
2955 Campus Drive, Suite 400
San Mateo, CA 94403
Phone: 650-295-7600
Fax: 650-295-7700
http://www.rsa.com

Security Dynamics Technologies, Inc.
20 Crosby Drive
Bedford, MA 01730
Phone: 781-687-7800
Fax: 781-687-7010
http://www.securitydynamics.com

INTRODUCTION

Intel and RSA Data Security recently announced that they would work together to improve computer security via RSA Data Security software and Intel hardware. Intel, the world-leading microprocessor manufacturer, announced that the hardware-based *Intel® Random Number Generator* (RNG) would be a standard part of their next-generation 8XX series chipsets. RSA Data Security, the world leader in security software, announced that their development tools and security software products would utilize this hardware capability to make applications more secure through *cryptography*.

Cryptography is the science of *encrypting*, or scrambling, data to protect it and then *decrypting*, or unscrambling, data to make it intelligible to the user or application. Central to cryptography are *keys*, small, unique pieces of data which function much as physical keys do for locks. That is, only the correct key will "unlock" a given piece of encrypted data.

There are two kinds of cryptography in common use: *symmetric* and *asymmetric*. In symmetric cryptography, the key that decrypts the data is identical to the one that encrypted it. Conceptually, symmetric cryptography is simple: One simply distributes a key to two or more communicating parties, and they begin using it to encrypt and decrypt. In practice, symmetric cryptography raises a "chicken and egg" problem: Since keys must be kept secret, they must be distributed securely. But how can organizations implement strong security for key distribution, when strong security depends on having keys?

The most popular solution to the problem of key distribution is *asymmetric* or *public key* cryptography, such as cryptography based on the RSA algorithm. (An *algorithm* is a mathematical procedure for accomplishing a particular goal. An encryption algorithm defines a procedure for encrypting and decrypting data. Thus, an encryption algorithm is analogous to a lock, into which a key is inserted.)

With public key cryptography, keys come in pairs, with one *public* and one *private* key in each key pair. Public keys are deliberately made widely available, while private keys are secret. Thus, the public key can be transferred over non-secure networks. The private key, on the other hand, can be generated on the machine that needs to use it; it does not have to be transferred at all.

Data encrypted with the public key can only be decrypted with the private key, and vice versa. For example, a user's public key can be used to encrypt mail sent to that user. Then, the mail can only be decrypted with the user's private key. Since the public key is widely known, anyone can send secure mail to the user. However, only the user can read his or her mail, because only the user has the necessary private key. Public key cryptography can be used to encrypt session keys when they are distributed.

Symmetric keys are created using small bits of random data called *seeds* as raw material. The quality of the seed material (specifically, how random it is) determines the quality of the keys. Thus, independent software vendors (ISVs) typically go to some lengths to insure that seed material is highly random. Unfortunately, the process of generating seed material using only standard PC hardware is often laborious for the end user, as well as inefficient in that it takes a large number of processor cycles. In addition, the quality of the resulting seeds will vary with the level of programming skill.

The hardware-based RNG supports the symmetric key generation by providing a source of highly random numbers, which are used as seed material. The hardware RNG eliminates the need for user involvement in the seed generation process. It makes the process of seed generation highly efficient. And the quality of the seeds does not depend on a programmer's skills.

The hardware-based RNG is particularly important when a server, such as a Web server or file server, must perform encryption. Servers are generally unattended. Therefore, it is of great value to have a source of seed material which does not require user intervention.

The most important type of symmetric key is the *session key*, which is used only during a particular connection. In this case, it is the speed of the hardware RNG which is critical. Speed is important when creating session keys, because these keys are created on the fly when a connection is set up. Any delay in creating a session key delays the use of the connection. Typically, both parties involved in the session (or all parties, if more than two are involved) provide seed material for the session key. Thus, the slowest participant will frequently determine how quickly all the required seed material is available. For this reason, it is often desirable for all participants in a session to have a hardware RNG.

Random seeds are not typically required for generating asymmetric keys. For instance, an RSA key is created by multiplying two large prime numbers. In addition, asymmetric keys are typically generated offline and less often that symmetric keys. Thus, efficiency is not as critical for asymmetric keys as for symmetric keys. The hardware RNG is not involved in the creation of RSA asymmetric keys.

RSA software provides flexibility by supporting both Pentium III® and Celeron™ systems, with or without a hardware-based RNG. RSA software also expands a small amount of random seed material from the Intel RNG to produce multiple keys efficiently. Thus, the combination of hardware and software results in a highly efficient, reliable and flexible process for producing symmetric encryption keys.

THE RNG COMES TO THE FORE

The average person has probably never heard any discussion of random number generation prior to the announcement by Intel and RSA Data Security. This issue is coming to the fore now because of increasing security requirements, often associated with the growth of the Internet. In particular, as explained above, random numbers are a basic building block for cryptography, which in turn is a foundational technology for security. As security becomes more important, cryptography becomes more pervasive and complex, and the need for improved random number generation increases.

The Role of Random Numbers in Cryptography

The hardware-based RNG is used at the very first stage of producing encryption keys, namely, generating random seed material. A software-based pseudo random number generator (PRNG) expands and processes the seed material to produce a series of random numbers. Finally, using one or more of those random numbers as input, a software-based key generation algorithm produces keys. The hardware RNG is critical to this

process because it produces highly random numbers, and it is extremely important that the seed material used for keys be highly random.

Random numbers are series of numbers that exhibit no predictable patterns. For instance, tossing a perfectly constructed six-sided die a number of times generates a series of numbers between 1 and 6, in no predictable order, with each number having an equal probability of appearing on any given toss. Thus, tossing a perfect die generates a series of random numbers.

A truly random series, or *pure* random series, such as that produced by tossing a perfect die, would never exhibit any pattern no matter how much statistical analysis was applied to it. Usually, pure random numbers depend on observing some physical phenomenon, such as tosses of dice or the movement of electrons.

A *pseudo random number* series, in contrast, is usually produced using an algorithm. Typically, PRNGs are implemented in software, which is much more cost-effective and thousands of times faster than hardware for this purpose. The PRNG typically expands a small amount of seed data into a series of pseudo random numbers. The level of randomness of the series depends on the level of randomness of the seeds which are used as input to the algorithm. Highly random seeds yield series of pseudo random numbers which also have a high level of randomness. If the input to a PRNG is truly random, then the output of the PRNG can be considered truly random.

Seeds used to produce keys must be highly random, since unpredictability is a necessary quality in keys. Within the pool of numbers from which keys are chosen, there should be no way for an attacker to determine which keys will actually be used. For example, RSA keys are always products of two prime numbers. From within the pool of "products of two primes," there should be no way for an attacker to tell which number will actually be used as a key in any given instance. To produce such high quality keys, key generation software must start with highly random seed material.

Sources of Seed Material

There are two common approaches to producing seed material for computers: One is based on a specialized hardware-based RNG. The other uses standard hardware such as a keyboard or mouse.

Specialized hardware-based sources of random numbers have been available for systems for some time, usually implemented as special add-in cards. Such cards produce truly random numbers S. However, there has been no standard for accessing such hardware, and the cards have been expensive. Thus, these solutions have not been widely implemented.

Some computers will have specialized RNG hardware, but most will not. Thus, software-based PRNGs should be able to get seed material from a hardware-based RNG if it is available, but should not require a hardware-based RNG.

In addition, when a hardware-based RNG is available, the software-based PRNG should take a relatively small amount of seed material from the hardware-based RNG and expand it to provide the random numbers required by the key generation software. This approach has the advantage that the hardware-based RNG is used only at the beginning of the key generation process. Thus, multiple applications can share a hardware-based RNG without overburdening it or being forced to wait in line. Demanding applications, such as busy Web servers using the hardware-based RNG, do not have to continually access the RNG, which could slow them down. Instead, they get a small amount of seed material and the PRNG uses it to create many keys very efficiently in software. Even if the hardware-based RNG becomes unavailable for some reason, the PRNG can continue producing keys once it has a seed.

When specialized random-number generation hardware is not available, programmers have to go to some lengths to generate seed material using standard computer hardware. For instance, the user may be required to make random motions with a mouse. The program translates the data on the mouse port into a series of numbers. Those numbers are then used as an ingredient for seed material that is fed into PRNG software. This approach has been used successfully and securely for years. However, it entails significant inconvenience for the end-user creating the seed material: Often, the end user must make mouse movements for several minutes to produce sufficient seed material. In addition, the seeds are not highly random. Thus, developers have to create more complex PRNGs that can compensate for expected non-random characteristics of seeds. Another disadvantage of this approach is that it depends on a user at the machine to generate random numbers. Servers are typically unattended, and there may be no user available when the server needs to generate keys. Thus, unattended servers are particularly appropriate platforms for hardware-based RNGs.

Random number generation hardware from Intel, combined with RSA Data Security software, provides an inexpensive, ubiquitous, automated and streamlined process for generating seed material. In addition, the seed material produced is truly random.

Let's look at these advantages one at a time:

- *Inexpensive:*
 Large-scale manufacturing lowers costs.

- *Ubiquitous:*
 Intel will incorporate its hardware RNG on their new Pentium III and Celeron -based platforms. RSA will provide independent software vendors (ISVs) with the higher-level software tools that will provide convenient access to the Intel RNG. Knowing that both the hardware RNG and the software required to integrate the RNG into applications will be ubiquitous, ISVs can target development around this feature, secure in the knowledge that they will reach a great majority of the market.

- *Automated:*
 From the point of view of the person who has to create keys (usually the end user of an application), the most obvious advantage of hardware-based random number generation will be increased automation. In particular, the user will no longer have to perform actions like making random movements with a mouse for a number of minutes. This is particularly important for servers, which are often unattended. In addition to being more convenient, the automated procedure is more secure, since it eliminates any possibility of the end user not following instructions or attempting to circumvent proper procedure.

- *Randomness:*
 Seeds produced using methods such as mouse movements are not as random as seeds produced by a hardware-based RNG. The output of the hardware-based Intel RNG is truly random.

- *Streamlined:*
 With hardware RNGs, PRNGs can be simpler and faster, since they do not need to compensate for non-random characteristics of seeds. Developers benefit in that PRNGs are easier to program. Users benefit because PRNGs operate more efficiently.

THE DEMAND

The convenience and low cost of hardware-based RNGs has become more important as security has become more pervasive and complex.

Early on, encryption was used only in a few very high security applications, and by a relatively small number of users. Today, the security requirements of the Internet have brought encryption to a wide variety of commercial applications, including e-mail, Web access and virtual private networks (VPNs). Applications developed in-house are increasingly being enabled with encryption, as well, since they often deal with sensitive corporate data.

For instance, e-mail is carrying not only memos and personal notes, but contracts, bids, and sensitive financial information. The Web is being used not only for publishing corporate brochures but also for software distribution and e-commerce. Through VPNs, the Internet is becoming an extension of corporate networks in addition to a shared public network.

Secure e-mail, Web access, e-commerce, and VPNs require strong security, including session encryption, to protect communications, content and e-commerce transactions. Thus, the growth of these Internet-oriented applications means that more companies and individuals are using more cryptography. More programmers are also required to integrate encryption into their applications.

For instance, e-mail and messaging products now typically support Secure Multipurpose Internet Mail Extensions (S/MIME). S/MIME supports symmetric encryption based on RC2, Digital Encryption Standard (DES) or Triple-DES.

Similarly, Web browsers and servers use symmetric encryption for confidentiality, for applications like online banking and online shopping. In these applications, servers typically authenticate themselves to clients using the Secure Sockets Layer (SSL) protocol. SSL also encrypts traffic during transmission using the RC4 symmetric encryption algorithm. SSL supports Hypertext Transfer Protocol (HTTP), as well as other protocols such as File Transfer Protocol (FTP) and Telnet.

Encryption, either symmetric or asymmetric, is also the main technology used to convert standard Internet links into VPNs, either for site-to-site privacy (router-to-router) or for secure remote access (client-to-server). VPNs are based on protocols such as Point-to-Point Protocol (PPP), IP Security (IPSEC), Microsoft's Point-to-Point Tunneling Protocol (PPTP), Cisco's Layer Two Forwarding (L2F), and the emerging Layer 2 Tunneling Protocol (L2TP), which combines the best features of PPTP and L2F. With any of these protocols, the term "VPN" implies encryption. For instance, PPTP uses RSA asymmetric cryptography and DES symmetric cryptography. IPSEC uses either RSA or Diffie-Hellman asymmetric cryptography. For symmetric cryptography in IPSEC, DES and Triple-DES are commonly used, with RC5 frequently being brought in when faster and stronger security is required.

All these applications are experiencing rapid growth, increasing the number of keys in use. As security requirements increase, companies and individuals also update keys more frequently, taking the old keys out of service and issuing new ones. This, too, increases the rate at which keys must be produced. As the computing power available to attackers grows, companies must use longer keys, as well.

More keys, longer keys, and more frequent updates all make the efficiency of key generation a more critical concern.

FOILING ATTACKS

When attempting to break an encryption system, perpetrators have two basic lines of attack at their disposal: "cracking" the algorithm or obtaining keys.

Cracking the Algorithm

On the one hand, attackers can attempt to "crack" the algorithm. That is, they can attempt to find a mathematical means of decrypting all material encrypted with that algorithm, without using a key. Naturally, encryption algorithms are designed to make such "lock-picking" impossible. However, high-level hackers, such as mathematicians working for foreign governments , can examine algorithms, looking for flaws which they can exploit.

Success in this endeavor gives access to everything encrypted with that algorithm, regardless of the key used. However, it is also an endeavor that can consume huge amounts of time without yielding any results. The best defense against this type of attack is a well-designed, well-studied, time-tested algorithm.

Obtaining Keys

The second basic mode of attack is to obtain one or more keys and decrypt material using a key. A key may be stolen, for instance, by retrieving it from a hard disk, if it is stored without proper security. Or, a key may be guessed by *brute force*. That is, the attacker can simply try all possible keys, one after another, until one is found that decrypts the desired material. For instance, in January 1999, in response to RSA Data Security's "DES Challenge," a DES key was discovered by brute force in 21 hours.

Brute force can also be used in an attempt to guess the seed material that was used to produce a series of keys. Success in guessing a seed gives the attacker all the keys in that series.

Whereas cracking an algorithm is an historic event, keys are stolen every day, often due to poorly designed security software, lax policies or careless users. Stealing a key gives access to a much more limited range of information than cracking an algorithm, because a key is usually associated with a single user and a single application. Sometimes a key is associated only with a particular function within an application. For instance, a user may have one key for encrypting files and email, and another for digital signing. Keys also have limited lifetimes. For example, most companies give users new keys at regular intervals. Some protocols automatically limit the lifetimes of the keys they use. For instance, symmetric SSL keys are only good during one session. For each connection, SSL uses a different symmetric key.

Despite the fact that keys are limited in their scope, stealing a key should be for all practical purposes impossible in a properly designed and implemented security system. One aspect of

this is developing appropriate security procedures and training people to use them.

If well-designed security policies and vigilant users make it impossible for an attacker to steal keys, the attacker has only the brute force attack to fall back on. Hardware-based random number generation, combined with RSA Data Security software, makes it easier for developers to create security software that is more immune to brute force attacks. In particular, hardware-based random number generation makes it easier to create highly random seeds in a secure fashion. Highly random seeds result in highly random keys. If keys are highly random, the attacker will not be able to eliminate any keys from consideration. The more possible keys, the lower the probability of the attacker guessing the right key. Given long enough keys and assumed limits on time and computing power available to the attacker, the probability can be reduced to negligible proportions.

The Value of Long Seeds and Keys

In general, the longer a key or a seed is, the harder it is to guess by brute force. For instance:

- If a key were only 8 bits in length, there would be only 256 possibilities to try, and the correct key could be guessed, on average, in 128 attempts.

- If a key is 128 bits in length, there are 340,000,000,000,000,000,000,000,000,000,000,000 possibilities, and it takes 170,000,000,000,000,000,000,000,000,000,000,000 attempts, on average, to guess the correct key.

Making more attempts will either take more time or use more computing power, or both. The security system must implement keys and seeds that are so long that the time and / or computing power required to guess a key or seed by brute force is prohibitive. The current recommended length for RSA Data Security asymmetric keys, for instance, is 1024 bits. There are no recorded instances of an asymmetric key of that length being guessed. For symmetric keys, 128 bits is currently considered a safe length.

A PRNG expands seed material to create random numbers which are used as input to key generation algorithms. Multiple large random numbers may be needed to create a key. Thus, many large random numbers may be required to produce a series of keys. A PRNG needs abundant seed material in order to create many long random numbers. A hardware-based RNG produces a continual stream of random data, providing abundant seed material. In comparison, mouse movements or keystrokes are very intermittent. Seed material that might take minutes to produce using a mouse could be created in a fraction of a second using a hardware RNG.

The Value of Randomness

If keys or seeds are totally random, a perpetrator has to try every possible combination of bits in a brute force attack. To the extent that keys or seeds are not completely random, the number of attempts required to guess a key or seed can be reduced. For instance, if a perpetrator knows that seed material is created by having the user hit random keys on a keyboard, the perpetrator can eliminate characters that are not on the keyboard. From experience, an attacker may even know that some keys are more likely to be hit than others. Similarly, if an attacker knows from experience that a key or seed will never contain more than five binary zeroes in a row, many keys or seeds can be eliminated as possibilities. Perpetrators can discover such non-random characteristics of keys and seeds by examining the output of a key generator or seed generator over a period of time.

Specialized hardware produces seeds that are more random than seeds produced using standard hardware such as a mouse. Highly random seeds are hard to guess. Highly random seeds are also necessary to produce highly random, hard-to-guess keys.

SEED CREATION: RIGHT AND WRONG

With skilled programmers and vigilant users, highly random keys can be produced without a hardware-based RNG, as RSA Data Security has demonstrated for many years. However, a hardware-based RNG eliminates a number of possible weak points in software-based seed generation.

Potential Trouble Spots for Software-Based Seed Generation

There are three main potential trouble spots for software-based seed generation:
1) Insecure programming techniques
2) Programmer error
3) Lax users

Insecure Programming Techniques

As an example of an insecure programming technique, some pseudo RNGs use arithmetic functions to produce seeds. There are many functions that produce very long sequences of numbers before non-random patterns become evident. Such arithmetic functions are tempting sources of seed material. However, if a hacker guesses or (more likely) is told which arithmetic function is being used, seed material becomes much more predictable, making keys equally predictable.

Another tempting source for seed material is the computer's clock. The assumption here is that the perpetrator cannot guess or control the time and date when seed material is gathered. The output of the clock, looked at on its own, is anything but random. However, from the point of view of a perpetrator, it may be hard to predict.

Using only clock output as a seed is not a good idea, however, because it has many known characteristics, and a perpetrator may be able to predict certain repeating patterns. In addition, a determined perpetrator might be able to modify the security software to reset the clock when seed material is gathered, thus producing known seed material. This would permit the perpetrator to predict all the resulting keys.

Programmer Error

One viable approach to getting seed material is to use a hardware source, such as keystrokes or mouse movements, that exhibits a degree of true randomness. Even then, these sources are not completely random. For instance, keystrokes are limited to the characters on the keyboard, and a typical user will type some keys more frequently than others.

To create better seed material, developers typically combine a number of different inputs, such as the computer clock, the contents of disk files, mouse movements and recent keystrokes, as

well as complex arithmetic functions. The complexity of the method used to create the seed material makes it much more difficult to guess. However, even with combined inputs, seeds are not completely random, and further processing may be required to increase the randomness of the seed material. As more programmers begin to implement key generation software, it becomes increasingly likely that inexperienced programmers will make mistakes which either create vulnerabilities in the security system or reduce the randomness of the seeds.

Lax Users

A requirement to type on the keyboard or move the mouse also makes the seed-generation process more time-consuming and tedious for the user. In an attempt to speed the process up, users may create vulnerabilities. For instance, a user asked to type random keys might simply hit the space bar repeatedly. Keys generated from such non-random seed material would also be non-random.

Hardware-Based RNGs Eliminate Potential Trouble Spots

Developers implementing software-only RNGs have to insure that the approach to seed generation is well thought-out and well implemented. Even then, they can only recommend proper use of the system; they cannot insure it.

A hardware-based RNG saves both programmers and end users time and trouble. In most cases, it is not feasible, even for a sophisticated and determined attacker, to predict or control the output of a well-designed hardware RNG. (In practice, "well-designed" usually means that the hardware RNG meets some predefined standard, such as the FIPS standard for hardware RNGs.) Furthermore, the hardware RNG does not require the person creating the keys to do anything to create seed material.

For programmers, hardware-based random number generation removes the temptation to take quick and easy, but not entirely safe, routes to seed generation, because it provides a method that is quick, easy and safe. It does not depend on a programmer going to the proper lengths to create good seed material.

How Good is "Good Enough"?

Exactly how long and how random do keys or seeds need to be? Perpetrators, like everyone else, have access to computer systems that, with each succeeding generation, get faster and more powerful at an exponential rate. Thus, security systems must implement progressively longer and / or more random seeds and keys as time goes on. The "impossible-to-guess" seed or key is a moving target. Security vendors must err on the side of caution, creating seeds and keys that are impossible to guess by brute force with today's technology, and which will remain impossible to guess for the foreseeable future. RSA recommends that the seed be at least the same length as the desired keys.

SUMMING UP: HOW HARDWARE HELPS

Security vendors such as RSA Data Security have produced high-quality symmetric keys for years without hardware-based RNGs in computers. In order to do this, they have developed techniques for producing seed material without specialized hardware. When properly implemented, these techniques result in keys which are highly random and thus resistant to brute force attacks. Sufficiently complex methods of generating seed material have also prevented hackers from guessing the seed itself. As a result, for instance, RSA Data Security's PRNG has never been known to be cracked, nor has it been considered a hopeful point of attack.

However, a hardware-based RNG has the following advantages:

- It eliminates the need for programmers to compensate for non-random characteristics of seed material.
- It produces truly random numbers, so there is never a question whether the job has been done right. Because seed material is more trusted, keys are also more trusted.
- It saves the person creating the seed from having to move a mouse randomly or perform other tedious actions.

To these advantages, a well-designed and well-implemented PRNG adds the following advantages:

- It offers flexibility. The security software can work with or without a hardware-based PRNG.
- It expands seed material, efficiently producing many keys much faster than could be done in hardware alone. This efficiency makes it possible for multiple applications to share a single hardware RNG without having to wait for one another.
- It increases reliability. If the hardware RNG becomes unavailable for any reason, keys can still be produced, either using seed material based on non-specialized hardware, or using seed material already gathered from the hardware RNG.

RSA DATA SECURITY PRODUCTS

In June 1999, the hardware-based Intel Random Number Generator will be supported in RSA's BSAFE® Crypto-C and BSAFE Crypto-J software development kits (SDKs). BSAFE Crypto-C is the world's most popular cryptographic SDK for the C programming language, with the widest range of data encryption and signing algorithms available today. It contains easy-to-use C language security tools and an encryption engine, allowing developers to integrate state-of-the art privacy and authentication features into virtually any application. BSAFE Crypto-J provides a 100% Pure Java certified version of the SDK, with the same level of functionality.

Developers who are already using these RSA Data Security products will find it very easy to incorporate hardware-based random number generation into their software products. For users, the change can be entirely transparent. Programs will be able to determine whether or not a hardware-based RNG is available on the PC platform and then make use of it if it is there.

Later this year, hardware-based random number generation will also be supported in BSAFE Cert-C certificate processing components for the C language, and in Cert-J, certificate processing components for Java. A digital certificate serves as a "digital ID" that notarizes the association between an RSA Data Security public key and its legitimate owner, in the same way that a driver's license proves identity.

For more information, contact RSA directly at (650) 295-7600, or visit RSA's Web site at http://www.rsa.com.

Copyright © 1999, RSA Data Security, Inc. All rights reserved. RSA and RSA Secure are trademarks of RSA Data Security, Inc. BSAFE is a registered trademark of RSA Data Security, Inc. Security Dynamics is a trademark of Security Dynamics Technologies, Inc. All other company and product names are the property of their respective owners.